US006292191B1

(12) **United States Patent**
Vaswani et al.

(10) **Patent No.: US 6,292,191 B1**
(45) **Date of Patent: *Sep. 18, 2001**

(54) **DYNAMICALLY SELECTABLE MIP MAP BLENDING FOR A SOFTWARE GRAPHICS ENGINE**

(75) Inventors: **Gautam P. Vaswani**, Austin; **Daniel Wilde**, Cedar Park, both of TX (US)

(73) Assignee: **Cirrus Logic, Inc.**, Austin, TX (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **08/976,523**

(22) Filed: **Nov. 21, 1997**

Related U.S. Application Data

(63) Continuation-in-part of application No. 08/965,381, filed on Oct. 23, 1997, and a continuation-in-part of application No. 08/949,177, filed on Oct. 10, 1997, application No. 08/976,523, which is a continuation-in-part of application No. 08/774,787, filed on Dec. 30, 1996, now Pat. No. 5,835,097.

(51) **Int. Cl.**[7] **G06T 11/40**
(52) **U.S. Cl.** **345/430**; 345/425; 345/439
(58) **Field of Search** 345/426, 428, 345/430, 431

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,651,104	*	7/1997	Cosman	345/428
5,734,386	*	3/1998	Cosman	345/430
5,786,822	*	7/1998	Sakaibara et al.	345/430
5,835,097	*	11/1998	Vaswani et al.	345/430
5,949,426	*	9/1999	Rich	345/430
5,953,015	*	9/1999	Choi	345/430

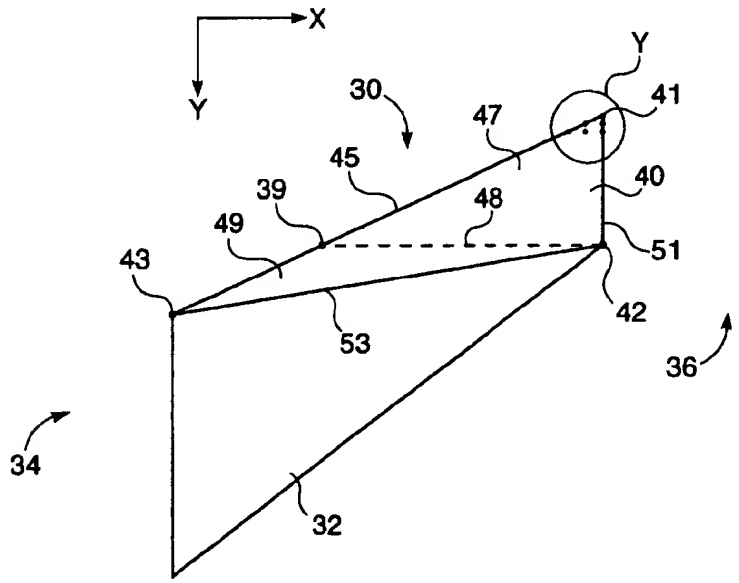
* cited by examiner

Primary Examiner—Mark Zimmerman
Assistant Examiner—Kimbinh T. Nguyen
(74) *Attorney, Agent, or Firm*—Jonathan M. Harrison; Steven Lin; Robert Platt Bell

(57) **ABSTRACT**

Graphics software renders polygons with texture using an improved MIP mapping technique in which texels from multiple MIP maps are blended together. The software renders the pixels in a polygon and selects texture elements ("texels") from an appropriate texture map to be applied to the pixels. The software further generates texel coordinate values to select texel values from a set of texture maps, each map varying from the others by the level of detail of the texture. The software then computes a scale factor for each texel value according an area bounded by adjacent texel coordinates. In one embodiment, vectors are defined for each the adjacent texels and the area is determined from the magnitude of the cross product of the vectors. The scale factor is then used to compute a weighted average of texels from one or more MIP maps. Further, for certain area values, no averaging occurs or, alternatively, the scale factor is set to 1.0.

16 Claims, 9 Drawing Sheets



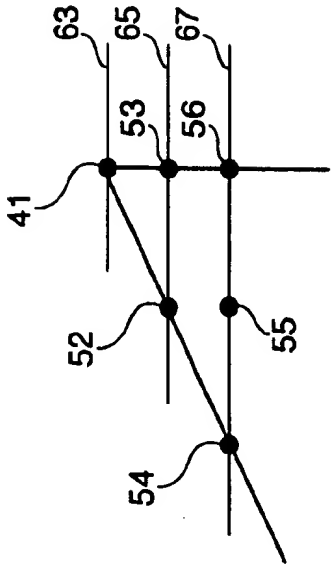


Figure 1a

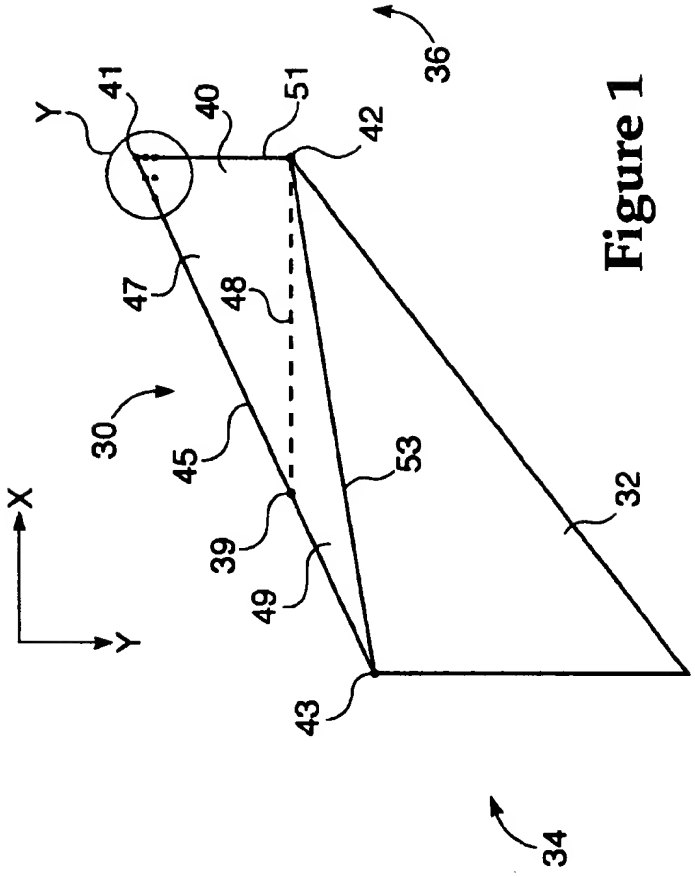


Figure 1

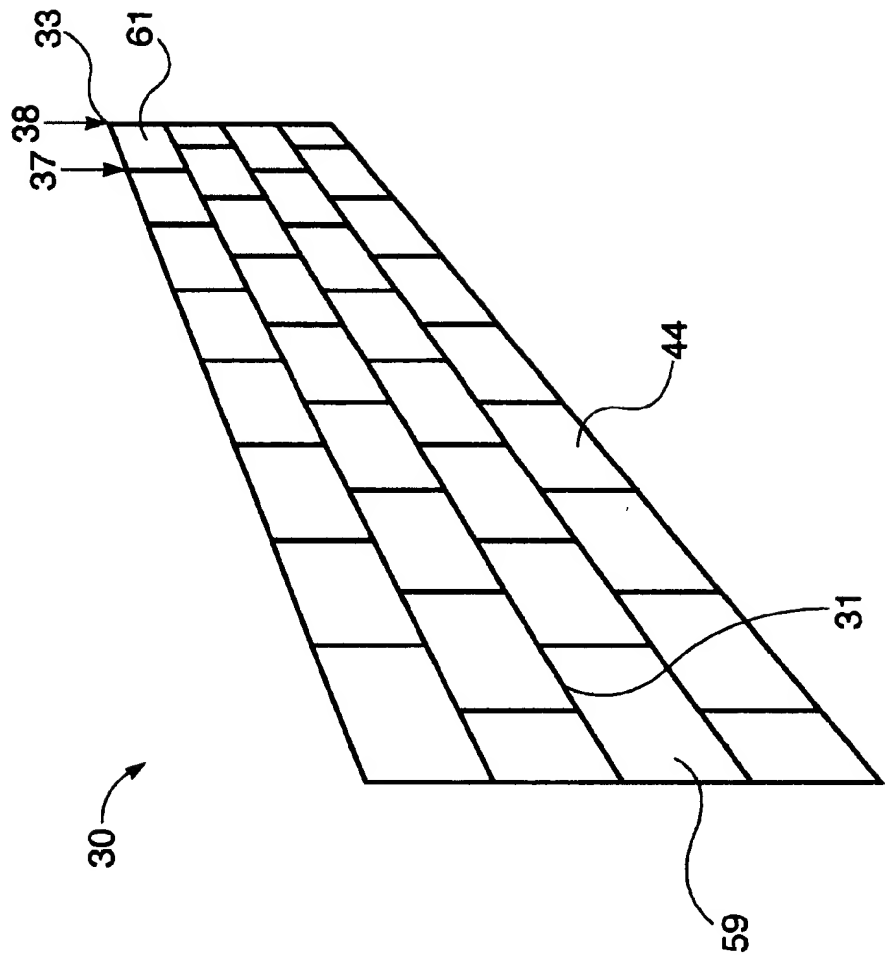


Figure 2

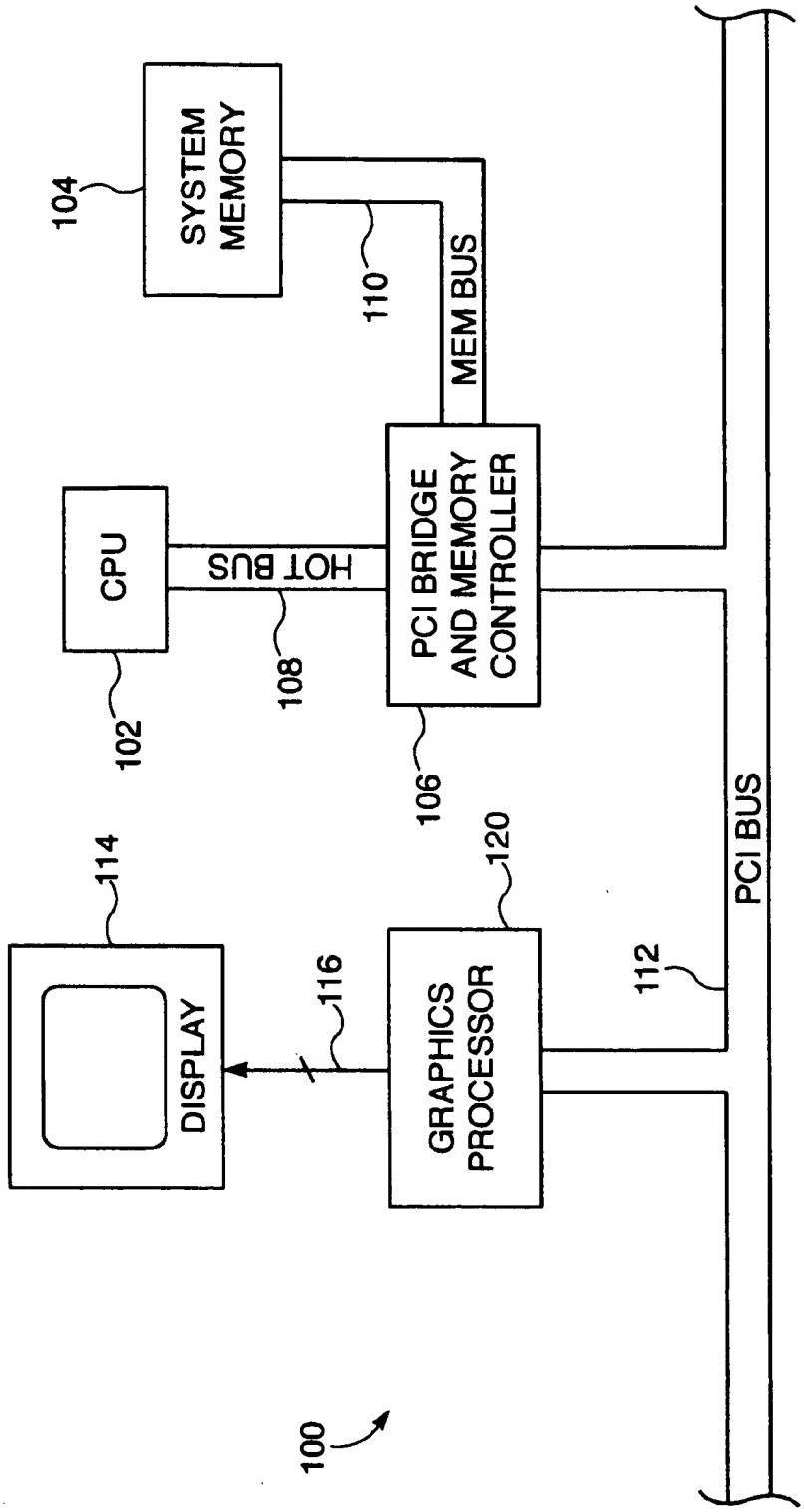

SYSTEM MEMORY
104
110
MEM BUS
CPU
HOT BUS
PCI BRIDGE AND MEMORY CONTROLLER
102
108
106
114
120
DISPLAY
116
GRAPHICS PROCESSOR
112
PCI BUS
100

Figure 3

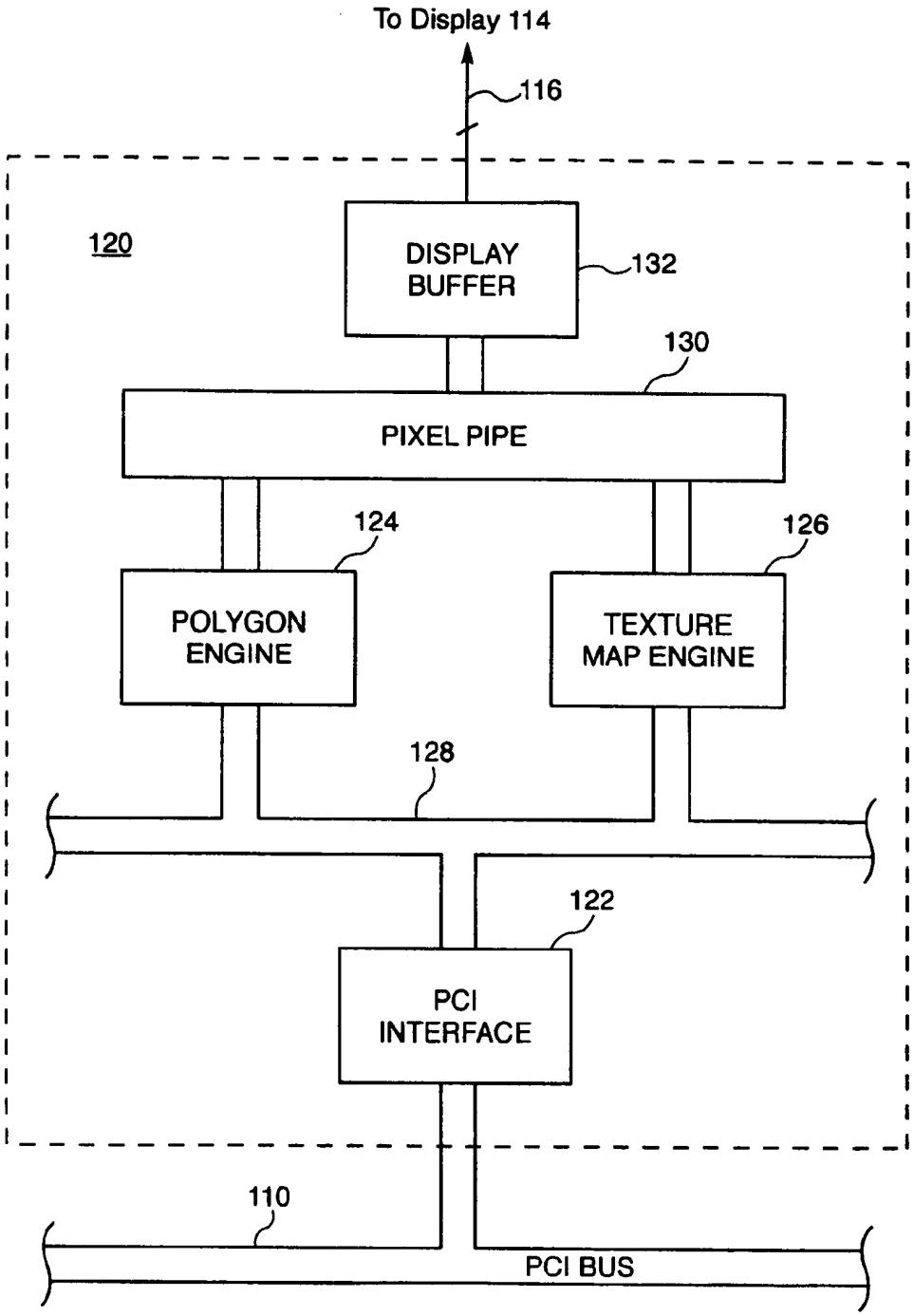

To Display 114
116
120
DISPLAY BUFFER
132
130
PIXEL PIPE
124
126
POLYGON ENGINE
TEXTURE MAP ENGINE
128
122
PCI INTERFACE
110
PCI BUS

Figure 4

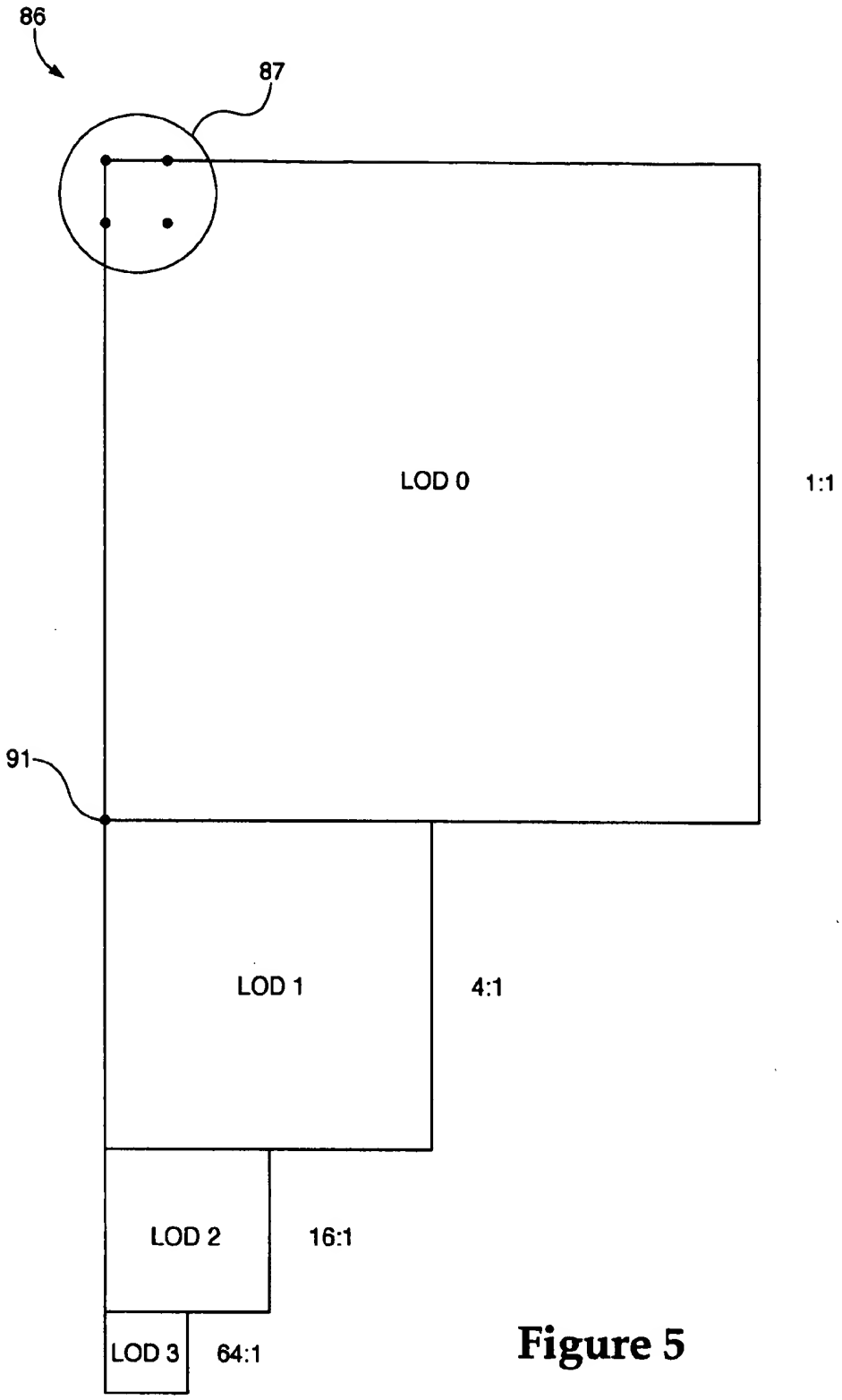


Figure 5

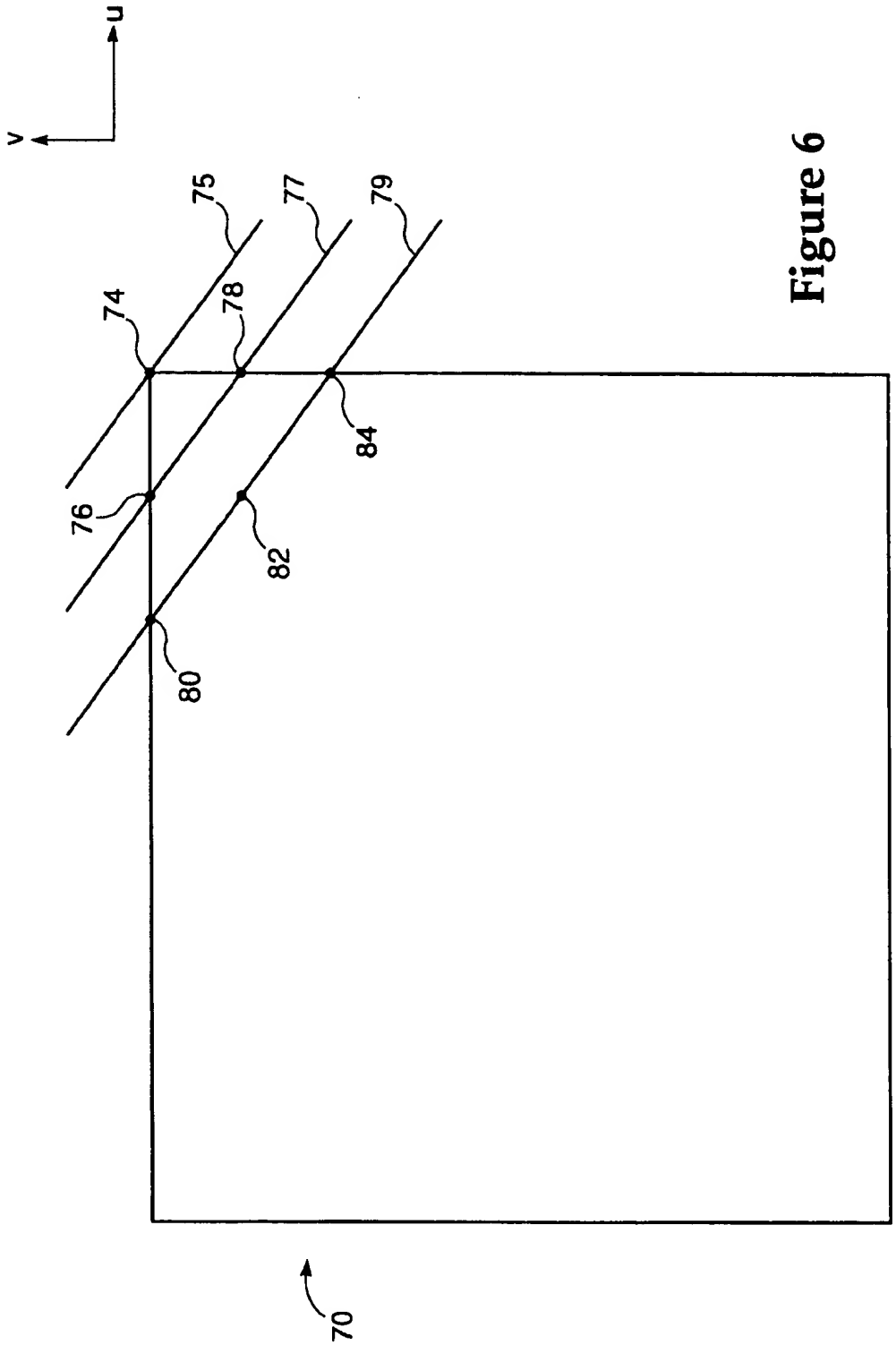


Figure 6

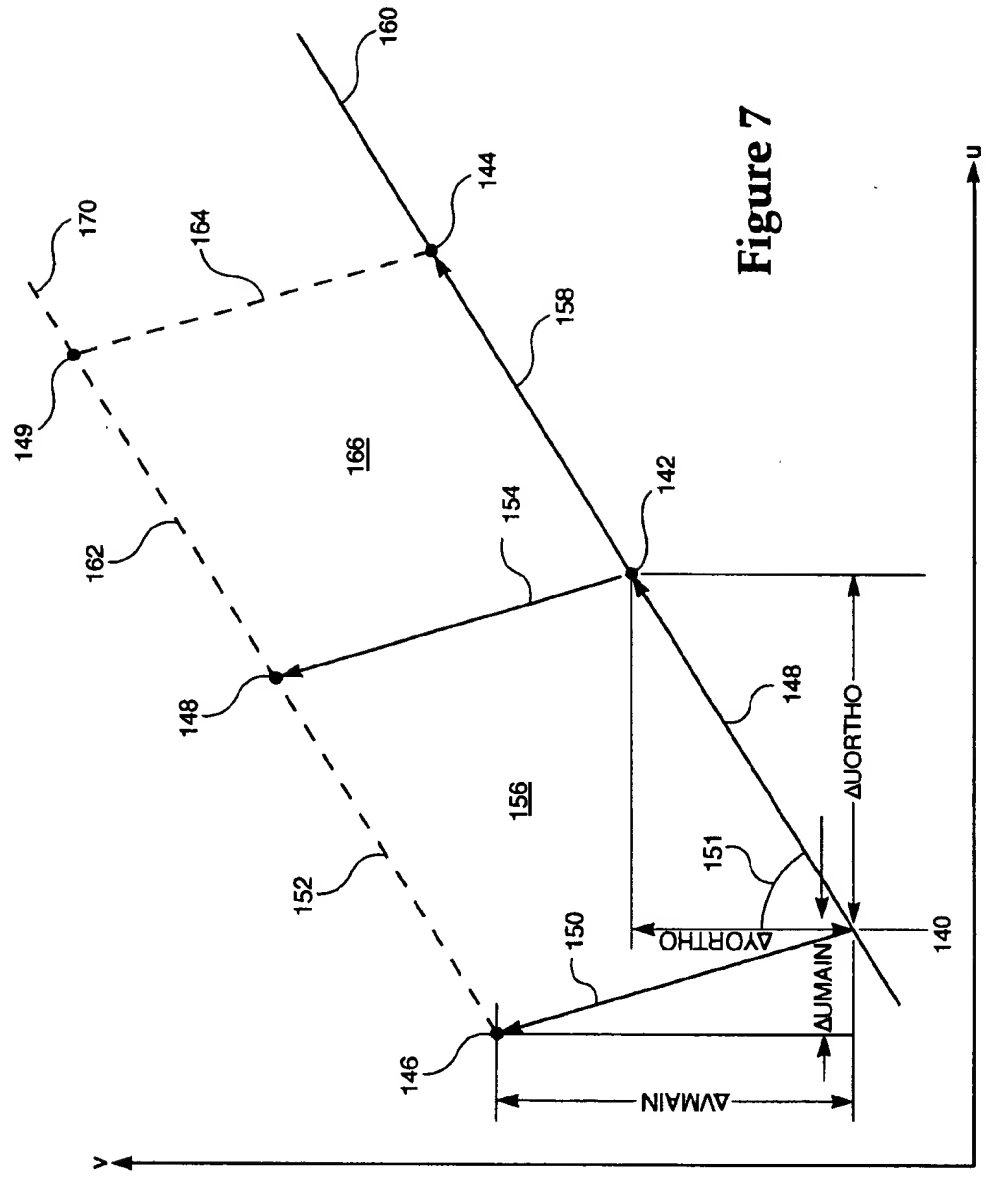

160
144
170
164
158
149
166
142
154
162
148
148
156
151
ΔUORTHO
152
150
140
ΔVORTHO
ΔUMAIN
146
ΔVMAIN
v
u

Figure 7

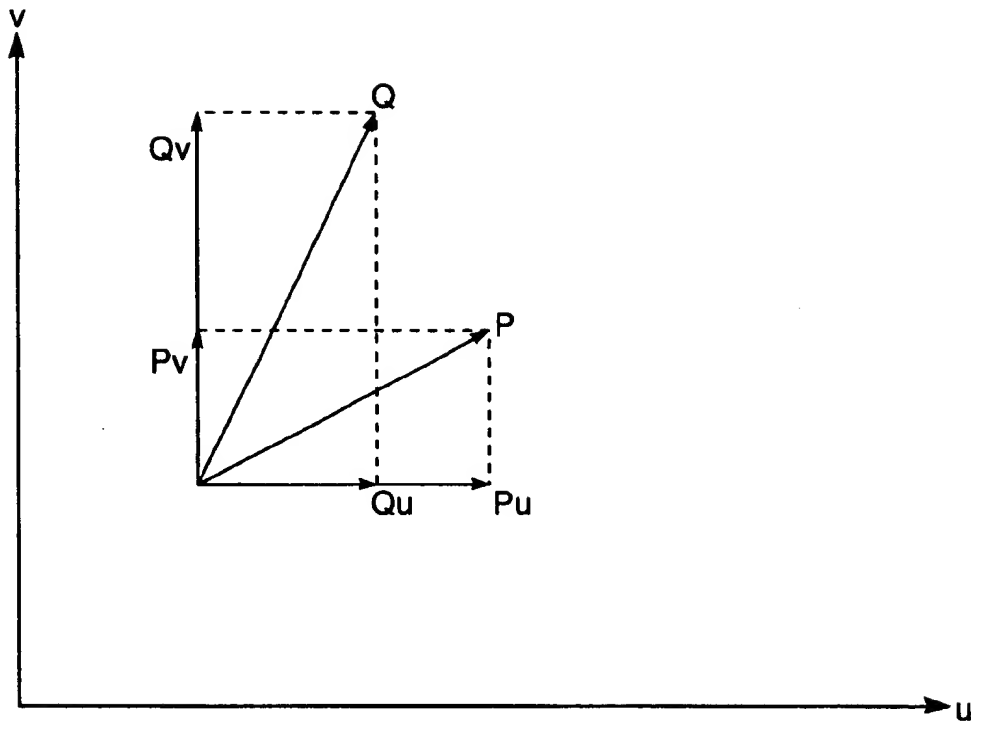


Figure 8

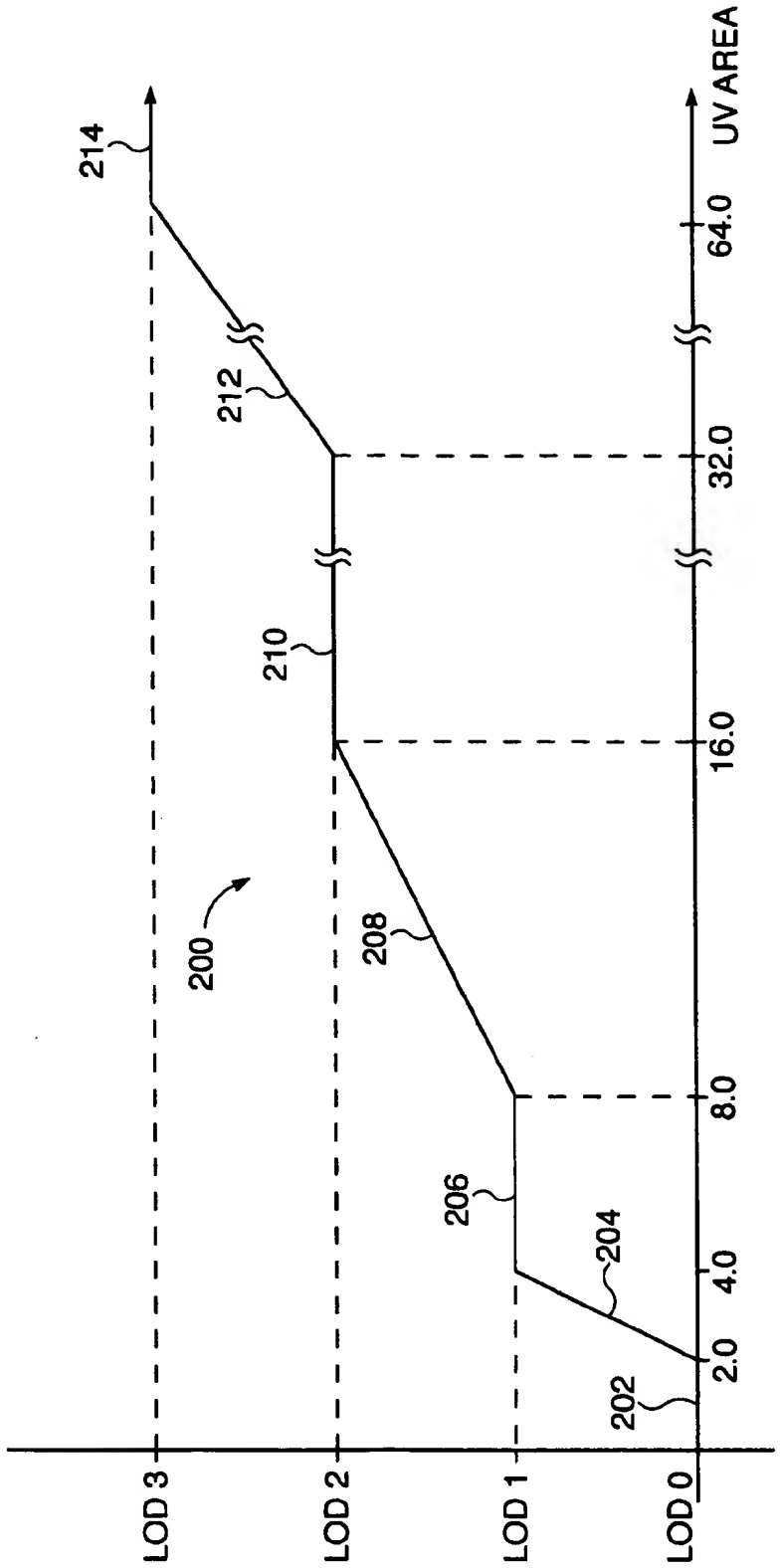

UV AREA
64.0
32.0
16.0
8.0
4.0
2.0
214
212
210
208
206
204
202
200
LOD 3
LOD 2
LOD 1
LOD 0

Figure 9

DYNAMICALLY SELECTABLE MIP MAP BLENDING FOR A SOFTWARE GRAPHICS ENGINE

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation-in-part of application Ser. No. 08/949177, filed Oct. 10, 1997, entitled "Improved MIP Map Blending in a Graphics Processor."

This application is also a Continuation-In-Part of U.S. patent application Ser. No. 08/774,787, filed Dec. 30, 1996, now U.S. Pat. No. 5,835,097 issued Nov. 10, 1998 and a Continuation-In-Part of U.S. patent application Ser. No. 08/965,381 filed Nov. 23, 1997.

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

Not applicable.

BACKGROUND OF THE INVENTION

A. Field of the Invention

The present invention relates generally to a graphics system for a personal computer. More particularly, the present invention relates to a method and apparatus for rendering polygons in a pixel grid. Still more particularly, the present invention relates to an improved method of applying texture to polygons.

B. Background of the Invention

Sophisticated graphics packages have been used for some time in expensive computer aided drafting, design and simulation systems. Increased capabilities of graphic controllers and display systems, combined with standardized graphics languages, have made complex graphics functions available in even the most routine applications. For example, word processors, spread sheets and desktop publishing packages now include relatively sophisticated graphics capabilities. Three-dimensional (3D) displays have become common in games, animation, and multimedia communication and drawing packages.

The availability of sophisticated graphics in PC's has driven a demand for even greater graphic capabilities. To obtain these capabilities, graphic systems must be capable of performing more sophisticated functions in less time to process greater amounts of graphical data required by modem software applications. In particular, there is a continuing need for improvements in software algorithms and hardware implementations to draw three-dimensional objects using full color, shading, texture mapping, and transparency blending.

The development of raster display systems has dramatically reduced the overall cost and increased the capabilities of graphic systems. In a raster display system, a set of orthogonal or horizontal scan lines, each comprising a row of pixels, forms an array or grid of pixels to represent the entire screen area. The screen preferably comprises a cathode ray tube (CRT), LCD display, or the like, capable of scanning the entire pixel grid at a relatively high rate to reduce flicker. The pixel data preferably is stored in a frame buffer comprising dynamic random access memories (DRAM's), or more preferably video RAMs (VRAM's), where each pixel is represented by one or more bits depending upon the desired resolution. In many graphics systems, for example, each pixel is drawn or "rendered" with 24 bits of color information (8 bits for red, 8 bits for green, 8 bits for blue). Typical display systems are capable of drawing screens with multiple colors at a variety of screen resolutions, including resolutions of 640 pixels×480 pixels, 800×600, 1024×768, 1280×1024, or even higher pixel value combinations, depending upon the software drivers and the hardware used.

Typically, a video controller scans and converts the pixel data in the frame buffer to provide control signals for the screen system. In particular, the video controller renders the screen pixels, typically from the top of the screen to the bottom and from left to right, converting pixel data into color intensity values for each pixel. In a color graphics system using a CRT, three separate beams are controlled for each of the primary colors, where the intensity of each of the beams is determined by the pixel value corresponding to the respective colors. A similar system is used for LCD displays.

Other improvements have been made in the hardware realm. Graphics processors and accelerators are available with software drivers that interface the host central processing unit (CPU) to the graphics processor. In general, the software driver receives information for drawing objects on the screen, calculates certain basic parameters associated with the objects and provides these parameters to the graphics processor. The software driver then sends a command for the graphics processor to draw the object into the frame buffer. A graphics processor may use interpolation techniques in which the fundamental information for the object to be drawn comprises a set of initial and incremental parameters. As explained below, the graphics processor loads or otherwise receives the initial parameters for rendering a first pixel, and then interpolates the remaining pixels in an object by using the incremented parameters until the object is complete.

An exemplary interpolation method for drawing a typical polygon is shown in FIG. **1** in which polygon **30** represents a wall extending from the foreground **34** to the background **36**. Polygon **30** can be subdivided into two triangles **32** and **40**. In general, however, a polygon is subdivided into as many triangles as are necessary to represent the polygon. For example, ten thousand triangles or more may be required to create a realistic looking image of a human face. Graphics objects typically are represented with a collection of triangles because triangles are simple geometric shapes that can be characterized with relatively few values.

Referring still to FIG. **1**, each triangle **32, 40** may be drawn in two portions, such as portions **47, 49** of triangle **40**. To render triangle **40**, for example, a software driver receives vertex information (including x, y coordinates of each vertex **41, 42, 43**) and identifies a main slope line **45** extending the vertical length (in the y direction) of triangle **40** between vertices **41** and **43**. The other two sides **51, 53** of triangle **40** are referred to as opposite slopes. The polygon **40** is interpolated using orthogonal (ORTHO) horizontal scan lines of pixels that extend from the main slope **45** to the opposite edges **51, 53**. ORTHO Scan lines **63, 65, 67** represent the first three scan lines for triangle **40**. The vertical or y parameter is used as a reference coordinate, so that they pixel value is preferably incremented (or decremented depending on whether the triangle is rendered from top to bottom or bottom to top) by one for each scan line. A value for the main slope **45** is calculated as an initial condition and is used to compute the x coordinate of the first pixel in each scan line (i.e., the pixels on of the main slope). The software driver also determines initial and incremental width values for the scan lines and intepolates the width rather than edge walking the opposite slopes. The interpolated width value is loaded into a counter and decremented for each pixel in the current scan line. When the width counter becomes zero or otherwise reaches terminal count, the counter asserts a terminal count signal indicating that the scan line is complete.

Using such interpolation techniques, each triangle **32, 40** is drawn one row or horizontal scan line of pixels at a time.

For each scan line of triangle **40** (such as scan lines **63**, **65**, **67**), pixels are rendered from the main slope **45** to the opposite edges **51**, **53**. In the first scan line **63**, vertex pixel **41** is rendered. In the second scan line **65**, pixels **52** then **53** are rendered. In the third scan line **67**, pixels **54**, **55**, **56** are rendered.

Graphics processors, such as the GD5464 manufactured by Cirrus Logic, are capable of applying texture to polygons through a process referred to as texture mapping. Texture mapping techniques generally apply a bitmapped texture image to a polygon on the screen. A texture map typically is a two dimensional array of texture elements ("texels") that define a texture such as a brick, a carpet design, the grain of wood or any other texture that might be applied to an object on a computer screen. Referring now to FIG. **2**, an exemplary brick texture **44** is applied to polygon **30** to create the appearance of a brick wall. The brick texture **44** is represented by a texture map. The texture map used in FIG. **2** may represent only a single brick which is applied repeatedly to polygon **30** to cover the entire polygon with bricks.

Using standard techniques, the texture map associated with the texture **44** is applied to polygon **30** in such a way to give the appearance of perspective to the brick wall of FIG. **2**. To further improve upon the realism of the brick wall, the bricks that are close to the viewer (brick **59** for example) preferably are shown with greater detail than the bricks in the distance (brick **61**). The detail includes the imperfections in the exposed surface of brick **59**, the mortar between bricks and any other detailed desired. Such detail normally could not be seen in the distant brick **61**. Accordingly, the graphics processor uses different versions of the brick texture map to account for differences in the level of detail in an object drawn with perspective. Each version is itself a texture map that includes a predetermined amount of detail. The texture map used to render brick **61** would not include the imperfections and other detail that could be seen in brick **59**. Thus, a particular texture, such as the brick of FIG. **2**, may include two or more texture maps representing the brick with varying levels of detail. MIP (multim in parvum) mapping is a technique by which the graphics processor selects a texture map with an appropriate level of detail (also referred to as a MIP map) to render a polygon. Using MIP mapping, the graphics processor switches between the various level of detail texture maps as the polygon is rendered.

The desire for sophisticated and realistic computer graphics requires the graphics hardware and software to deliver high performance. Performance becomes especially critical for computer games which typically include moving objects. Standard MIP mapping techniques for switching between various level of detail texture maps generally include computationally intensive operations and require numerous memory accesses that impair a graphics system's ability to deliver detailed, realistic, three-dimensional moving images. Further, known MIP mapping techniques often switch between MIP maps in such a way as to cause noticeable transitions in the texturing that is applied to the polygons.

Accordingly, it would be desirable to develop a graphics system that provides a MIP mapping scheme that requires relatively few memory accesses and computations. Further, it would be desirable to provide a graphics system that reduces or eliminates the transitions in texturing characteristic of many previous MIP mapping schemes. Such a system would provide increased performance while also providing high quality graphics.

SUMMARY OF THE INVENTION

A graphics system includes a graphics controller for rendering polygons with texture using an improved MIP mapping technique. The graphics controller includes a polygon engine for rendering the pixels in a polygon and a texture map engine for determining texture element ("texel") values from one or more texture maps to be applied to the pixels rendered by the polygon engine. The texture map engine generates texel coordinate values from pixel coordinate values provided by the polygon engine. A set of texture maps is used to represent a single texture with each texture map in the set representing a different level of detail of texture. The graphics controller either selects a texel value from one texture map or combines together two texel values from two texture maps based upon the area bounded by adjacent texels. The graphics controller determines a coordinate of a texel for every pixel coordinate in the polygon and calculates the area bounded by neighboring texel values by calculating the cross product of vectors connecting pairs of adjacent texel coordinates.

The graphics controller compares the calculated areas with ranges of values and selects a single texel value to render a pixel or averages two texel values together based on the range of values to which each area corresponds. Some ranges of area values require the graphics processor to select a single texel value from a texture map, while other ranges require averaging of texel values from two or more texture maps. If the area is within a range of values requiring averaging, the texture map engine computes a weighted average of texel values from two texture maps using the calculated area as a scale factor. The graphics processor calculates an area for each texel coordinate and selects a single texel value from a single texture map or averages two texel values together on a pixel-by-pixel basis while rendering the polygon.

An embodiment of the invention includes software that calculates the area bounded by adjacent texel coordinates and compares that area to a range of area values. The software then blends one or more texels together based on the area value. The software may run in the graphics processor or in a CPU that operates the host computer in which the graphics processor resides. The range of area values comprises a plurality of subranges. For certain subranges, the software selects and calculates a weighted average of two or more texel values from different texture maps. The weighted average is based on a scale factor using the area previously calculated. For other subranges, the software selects a texel from a single texture map.

Thus, the present invention comprises a combination of features and advantages which enable it to overcome various problems of prior devices and methods. The various characteristics described above, as well as other features, will be readily apparent to those skilled in the art upon reading the following detailed description of the preferred embodiments of the invention, and by referring to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

For certain subranges of area values, the software selects a single texel from one texture map.

A better understanding of the present invention can be obtained when the following detailed description of the preferred embodiment is considered in conjunction with the following drawings, in which:

FIG. **1** is an exemplary polygon representing a wall drawn with perspective in a graphics system;

FIG. **1A** is a blow-up of a portion of the wall of FIG. **1**;

FIG. **2** is the exemplary wall of FIG. **1** drawn with brick texture maps to create the appearance of a brick wall;

FIG. **3** is a general block diagram of a graphics system including a graphics processor in accordance with the preferred embodiment;

FIG. **4** is a more detailed schematic diagram of the graphics processor of FIG. **4** in accordance with the preferred embodiment;

FIG. **5** shows multiple texture maps with varying levels of detail used to render polygons;

FIG. **6** shows an exemplary texture map and the selection of texels used to render a polygon;

FIG. **7** shows the MIP mapping technique of the preferred embodiment to determine which of a set of texture maps with varying levels of detail to use when rendering portion of a polygon;

FIG. **8** is a graph showing exemplary vectors represented by their rectangular coordinates; and

FIG. **9** is a graph showing how texels from more than one texture map are blended together in accordance with the preferred embodiment.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

Referring now to FIG. **3**, a computer system **100** constructed in accordance with a preferred embodiment generally includes CPU **102**, system memory **104**, a peripheral computer interconnect ("PCI") bridge and memory controller **106**, a graphics processor **120**, and a display **114**. The CPU **102**, system memory **104**, PCI bridge **106** and display **114** preferably are known components. The CPU **102** may include any available processor such as the Pentium MMX®, K6®, or any other processor capable of operating a computer system in a fashion consistent with the preferred embodiment. The system memory **104** preferably includes standard dynamic random access memory ("DRAM"), synchronous DRAM, or any other suitable type of memory. The PCI bridge and memory controller may include any suitable off the shelf device or may be a custom design. Display **114** includes standard cathode ray tube ("CRT") displays, flat panel displays, or any other display device capable of displaying graphics.

The CPU **102** connects to the PCI bridge and memory controller **106** via a host bus **108** which includes address, data, and control lines for transferring data. The PCI bridge and memory controller **106** also connects to system memory **110** via a memory bus **110** which also includes address, data, and control lines suitable for transferring data between system memory **104** and PCI bridge and memory controller **108**. The CPU **102** may initiate read and write cycles to system memory **104** by way of host bus **108**, PCI bridge and memory controller **106**, and memory bus **110** according to known techniques.

A system bus **112**, preferably including a PCI bus, although other bus architectures are also acceptable, connects the PCI bridge and memory controller **106** to graphics processor **120**. It should be recognized by those of ordinary skill in the art that other devices besides those shown in FIG. **3** may also connect to the PCI bus **112**. Examples of other devices include extended memory cards, video cards, and network cards, video cards, and network cards. Further, other bus standards besides the PCI bus, such as the advanced graphics part ("AGP") bus may be used as the system bus **112**. If system bus **112** is implemented as an AGP bus, then the bridge and memory controller **106** is constructed so as to couple to an AGP bus. Graphics data in the form of a display list is transferred between CPU **102** and graphics processor **120** by way of host bus **108**, PCI bridge and memory controller **106**, and PCI bus **112**. Once graphics data is received by graphics processor **120** over the PCI bus **112**, the graphics processor manipulates the data to provide appropriate signals over lines **116** to display **114** for displaying objects on the display.

Referring now to FIGS. **1** and **4**, the graphics processor **120** of the preferred embodiment includes a PCI interface **122**, a polygon engine **124**, a texture map engine **126**, a pixel pipe **130**, and a display interface **132**. The PCI interface **122**, polygon engine **124**, and texture map engine **126** couple together via bus **128**. Pixel x,y addresses are provided over bus **128** to the texture map engine **126** and are used by texture map engine **126** to generate texture map addresses, as explained below. The polygon engine **124** and the texture map engine **126** couple to the pixel pipe **130**. The display interface **132** uses information provided to it from the pixel pipe **130** to provide appropriate signals to the display **114** over lines **116**.

The polygon engine **124** provides x,y pixel coordinates along with lighting and depth information to the pixel pipe **130**. Concurrently, the texture engine **126** calculates a u,v coordinate for each pixel and fetches the texel color value (or simply "texel") from the appropriate texture map. The pixel pipe **130** performs basic 3D operations on the pixel color, depth, and texel values to generate a final color value to be applied to the pixel. This final color value is stored in the display buffer **132**.

In accordance with a preferred embodiment, graphics processor **120** receives data in the form of a display list from the CPU **102** or system memory **104** via the PCI bus **112**. The display list is stored in a register file in graphics processor **120** or memory (not shown) directly coupled to the graphics processor **120**. The display list includes all information needed to draw a polygon. It is assumed each polygon includes an upper or main triangle (such as triangle **47** in FIG. **1**) abutting a lower or opposite triangle (such as triangle **49**). The values in the display list include the data needed to render both upper and lower triangles. Table I below includes an exemplary display list identifying the values that are included in the list (first column of Table I) and the description of each value (second column). References to X and Y values refer to the x, y coordinates of pixels on the display (referred to as x, y pixel space). References to U and V values refer to the coordinates of texels in a texture map which are identified as u,v coordinates. The u, v coordinate system of a texture map is referred to as u, v texture map space.

TABLE I

Display List.

NAME	DESCRIPTION
X	Initial x pixel coordinate
Y	Initial y pixel coordinate
R	Initial Red value for initial x, y pixel
G	Initial Green value for initial x, y pixel
B	Inital Blue value for initial x, y pixel
ΔX MAIN	Main slope value: this value is added to the initial x coordinate on each step in y to generate the x starting point for each new ORTHO scan line.

TABLE I-continued

Display List.

NAME	DESCRIPTION
Y COUNT	Top count: Bottom count concatenated. Determine the number of steps in y for the upper and lower portions of the triangle drawn.
X WIDTH MAIN	Initial upper width value. Width of the first ORTHO scan line in x of the upper (main) triangle
X WIDTH OPP	Initial bottom width value. Width of the first ORTHO scan line in x of the lower (opposite) triangle
ΔX WIDTH MAIN	Main width slope. This value is the amount by which the width of each scan line in x of the upper (main) triangle is adjusted on each step in y.
ΔX WIDTH OPP	Opposite width slope. This value is the amount by which the width of each scan line in x of the lower (opposite) triangle is adjusted on each step in y.
ΔR MAIN	Red main slope. This value is the amount by which the red color component start value for each scan line in x is adjusted on each step in y.
ΔG MAIN	Green main slope. This value is the amount by which the green color component start value for each scan line in x is adjusted on each step in y.
ΔB MAIN	Blue main slope. This value is the amount by which the blue color component start value for each scan line in x is adjusted on each step in y.
ΔR ORTHO	Red ORTHO slope. This value is the amount by which the red color component is adjusted for each step in x along a scan line.
ΔG ORTHO	Green ORTHO slope. This value is the amount by which the green color component is adjusted for each step in x along a scan line.
ΔB ORTHO	Blue ORTHO slope. This value is the amount by which the blue color component is adjusted for each step in x along a scan line.
Z	Initial z pixel coordinate.
ΔZ MAIN	Z main slope value. Added to z to generate starting z coordinate for each new scan line.
ΔZ ORTHO	Z ORTHO value. This value is the amount by which the z coordinate is adjusted along a scan line on each step in x.
V	Initial v coordinate of first texel address in texture map being used.
U	Initial u coordinate of first texel address in texture map being used.
ΔV MAIN	V main slope value. Amount by which the v texel coordinate start value is adjusted on each step in y.
ΔU MAIN	U main slope value. Amount by which the u texel coordinate start value is adjusted on each step in y.
ΔV ORTHO	V ORTHO slope value. Amount by which the v texel coordinate is adjusted on each step in x.
ΔU ORTHO	U ORTHO slope value. Amount by which the u texel coordinate is adjusted on each step in x.
ΔV2 MAIN	Amount by which the ΔV MAIN value is adjusted per main slope step.
ΔU2 MAIN	Amount by which the ΔU MAIN value is adjusted per main slope step.
ΔV2 ORTHO	Amount by which the ΔV ORTHO is adjusted per ortho step.
ΔU2 ORTHO	Amount by which the ΔU ORTHO is adjusted per ortho step.

It should be recognized that a display list may, and often will, include additional values. Thus, the values in the display list of Table I are exemplary only and are not exhaustive of all the values included in a typical display list.

The graphics processor **120** uses the values in the display list in Table I to draw a polygon and apply texture. How the graphics processor **120** renders a polygon with texture will now be described with reference to triangle **40** (FIG. 1). Referring to FIG. **1** and Table I, triangle **40** preferably is divided into two portions—an upper or main triangle **47** and a lower or opposite triangle **49** separated from main triangle **47** by dashed line **48**. For purposes of simplicity, the following discussion assumes the triangle **40** is drawn from top to bottom.

Triangle **40** is drawn in a series of horizontal ORTHO scan lines in which each scan line includes one or more pixels. Because the ORTHO scan lines are horizontal, only the x coordinate changes from one pixel in the scan line to the next. Further, the polygon engine **124** preferably increments the x coordinate by one as the graphics processor renders each pixel in succession along an ORTHO scan line. To draw the upper or main triangle **47**, the graphics processor needs to know or calculate the coordinate of the first pixel in the triangle (pixel **41**, for example), the coordinate of the first pixel in each ORTHO scan line, the number of pixels in each scan line, and the number of scan lines in the triangle. These values can be determined from the display list in Table I. The coordinate of the initial pixel is X, Y. The coordinate of the first pixel in each successive scan line is calculated by adding ΔX MAIN to X. It should be recognized that ΔX MAIN may be a positive or negative number depending on the slope of the line on which the first pixels in each ORTHO scan line lie. Thus, if the line on which the first pixels lie slopes down and to the left (as is the case for the main slope line **45** in FIG. **1**), then ΔX MAIN is a negative number because the x, y coordinate axes in the preferred embodiment have x coordinates increasing from left to right (y coordinates increase from the top of the display to the bottom). Thus, adding a negative value to an x coordinate produces an x coordinate to left on the display. Conversely, if the line on which the first pixels lie slopes down and to the right, ΔX MAIN will be a positive number. If the line is vertical, ΔX MAIN has a value of 0.

The number of pixels in each ORTHO scan line is calculated by adding ΔX WIDTH MAIN to the width (i. e., number of pixels in the x direction) of the previous scan line. For example, the ΔX WIDTH MAIN value for triangle **47** is 1. The width of the first scan line **63** (X WIDTH MAIN) is 1 because there is only 1 pixel (pixel **41**) in the first scan line **63**. Adding a ΔX WIDTH MAIN value of 1 to the X WIDTH MAIN value of 1 provides a width of the second scan line **65** of 2. Thus, there are 2 pixels in the second scan line (pixels **52**, **53**). The graphics processor then adds the ΔX WIDTH MAIN value of 1 to the width of the second scan

line (2 pixels) and thereby determines that there are 3 pixels in the third scan line **67** (pixels **54, 55, 56**). Drawing curvilinear shapes ("warping") can be achieved by varying ΔX WIDTH MAIN throughout the interpolation process.

The number of scan lines in a triangle is provided by the Y COUNT value which includes the number of scan lines in both the upper (main) triangle and lower (opposite) triangle. The portion of the Y COUNT value representing the number of scan lines in the main triangle preferably is loaded into a counter and decremented by one for each ORTHO scan line drawn. When the counter reaches its terminal count, the graphics processor has completed drawing the main triangle.

After the main triangle **47** is drawn, the opposite triangle **49** is then drawn using the same technique. The first pixel to be drawn preferably is pixel **39** and the width of the first scan line is the number of pixels along dashed line **48**.

In accordance with one preferred embodiment, the texture map engine **126** applies texels to each pixel in the triangles drawn by polygon engine **124**. The texels are stored in a texture map and are accessed using values in the display list in Table I above. Because polygons are often drawn with perspective, a set of texture maps in which each texture map has a particular level of detail that is different from the other texture maps in the set is used to render the polygon. Referring now to FIG. **5**, set of texture maps **86** preferably includes four levels of detail ("LOD")—LOD0, LOD1, LOD2, and LOD3. Each LOD is itself a texture map. All four LOD's are maps of the same basic texture differing only by the amount of detail in each map. LOD0 provides the most level of detail of any of the maps and is used for those portions of polygons that are close to viewer in which maximum detail is preferred (such as brick **59** in FIG. **2**). LOD1 provides the next most detail. LOD2 provides the next most level of detail after LOD0 and LOD1. Finally, LOD3 provides the least amount of detail and is used for distant portions of a polygon in which little detail is preferred (such as brick **61** in FIG. **2**). The LOD's may be any size desired. In accordance with a preferred embodiment, LOD0 includes 1024 rows and 1024 columns of texel values. Each texel value is represented by a combination of red, green, and blue color values and is one or more bytes long.

It should be apparent that in x, y pixel space polygons are drawn by providing color values to each and every pixel in an ORTHO scan line, incrementing the y coordinate to the next scan line and repeating the process. The texture map engine generates u,v texel coordinates. The texel from the texture map associated with the u,v coordinate is then selected to be applied to the x,y pixel coordinate. Any other scheme for selecting texels from a texture map, such as bilinear averaging, is also consistent with the preferred embodiment.

Unlike rendering pixels in x,y space in which each pixel is rendered, in u, v texture map space the texture map engine **126** may skip texels as it converts pixel coordinates to texel coordinates. Referring again to FIGS. **2** and **5**, the foreground brick **59** is rendered using LOD0 for maximum detail. Thus, all 1024×1024 (1M) texels in LOD0 may be used to render brick **59**. Brick **61** in the distant is drawn smaller than brick **59** to provide the illusion of perspective. Thus, there are fewer pixels used to render brick **61** than brick **59**. Because there may be many more texels in LOD0 than pixels in brick **61**, not all of the texels from LOD0 will be selected by the texture map engine **126** when converting x,y coordinate to u,v coordinates. It thus is preferable to use a reduced or compressed version of LOD0 to render brick **61**. LOD1–LOD3 provide compressed versions of LOD0, varying by the amount of compression. LOD1 preferably is one fourth of the size of LOD0 and includes **512** rows and **512** columns of texels. As such, LOD1 represents a four to one ("4:1") compression of LOD0. That is, LOD1 includes the same basic texture as LOD0 (albeit with less detail) in a texture map one-fourth the size of LOD0. The 4:1 compression is achieved by averaging four pixels in LOD0 (such as pixels **87** in FIG. **5**) to produce one pixel in LOD1, although other techniques for generating the LOD's are possible. LOD2 and LOD3 also include the same basic texture as LOD0 with varying degrees of compression. LOD2 includes **256** rows and columns of texels and thus is one sixteenth of the size of LOD0 (a compression of 16:1). Finally, LOD3 is one sixty-fourth of the size of LOD0 (64:1 compression) and includes 128 rows and columns of texels.

Selecting an appropriate LOD provides higher quality images and is more efficient than using a texture map with maximum detail. Using a texture map with more detail than that necessary for the polygon to be rendered often makes the polygon appear to sparkle resulting in an unrealistic appearance. Additionally, texels are normally read from system memory in a block of data that is stored in cache memory (not specifically shown) according to known techniques. In accordance with the preferred embodiment, computer system **100** reads an 8×8 block of 64 texels from system memory **104** when a texel value is desired and stores the 64 texel block in high speed cache memory. When the next texel is needed from the texture map, there is a substantial likelihood that the needed texel is one of the 64 texels already stored in cache memory (a cache "hit"). Accesses to cache memory occur at a much higher speed than accesses to standard system memory **104**, thereby speeding up the graphics process. If the next texel required is not one of the texels already stored in cache memory (a cache "miss"), a new 8×8 block of 64 texels must be read, and cached, to access the required texel value. Thus, using LOD0 when a smaller, more compressed LOD is appropriate slows the texture mapping process because a system memory access may be required for every texel needed. Instead, the graphics processor **120** increases the probability of a cache hit by using an appropriately sized LOD or pair of LOD's.

Referring now to FIGS. **1** and **6**, texture map engine **126** uses texels from an exemplary texture map **70** to apply to triangles **47, 49**. It should be apparent that selection of texels in u, v space from texture map **70** does not necessarily proceed along horizontal rows as is the case for rendering polygons in x, y space. For example, if texture map **70** is applied to polygon **40**, texel **74** may be the first texel selected to be applied to the first pixel rendered (pixel **41**). Further, texels **76, 78, 80, 82, 84** would be applied to pixels **52, 53, 54, 55, 56**, in order. Thus, ORTHO scan lines in u, v space (such as scan lines **75, 77, 79**) may lie at arbitrary angles.

Unlike polygon rendering in which only the x coordinate changes from one pixel to the next along an ORTHO scan line in x, y pixel space, both the u and v coordinate may change from one texel to the next along a single ORTHO scan line in texture space. The U and V values from Table I provide the coordinate of the first texel to be used from the texture map. The ΔU ORTHO and ΔV ORTHO values are used to calculate the u, v coordinate for each texel along an ORTHO scan line. Further, the ΔU2 ORTHO and ΔV2 ORTHO values are used to change the ΔU ORTHO and ΔV ORTHO values, allowing for perspective correct texture mapping. The ΔU ORTHO and ΔV ORTHO values are used to calculate the u, v coordinate for each texel along an ORTHO scan line. The ΔU MAIN and ΔV MAIN values are used to compute the u, v coordinate for the first texel in a particular ORTHO scan line. Perspective correct texture mapping is further implemented by using the ΔU2 MAIN and ΔV2 MAIN values to change the ΔU MAIN and ΔV MAIN values as the main slope is traversed. The MIP mapping technique of the present invention uses the ΔU

ORTHO, ΔV ORTHO, ΔU MAIN, ΔV MAIN values to blend together texel values from two or more LOD's as explained below.

Referring now to FIG. **7**, exemplary texels **140**,**142**, **144**, **146**, **148** from a texture map are shown in relation to the u and v axes. Texels **140**, **142**, **144** are texels along ORTHO scan line **160** and texels **146**, **148** are texels along ORTHO scan line **170**. In accordance with the interpolation technique explained above, the u,v address of texels and the delta values to the next texel in both the ortho and main slopes are available. Once the coordinate of texel **140** is determined by the texture map engine **126**, the coordinate of texel **142** can be calculated from the ΔU ORTHO and ΔV ORTHO values. Similarly, the ΔU MAIN and ΔV MAIN values are used to calculate the coordinate of texel **146** are known. In this manner, texel coordinate values for texels **144** and **148** can be calculated once texel coordinate **142**. Moreover, given one texel coordinate from a texture map, the two adjacent coordinates can be calculated given the ΔU ORTHO, ΔV ORTHO, ΔU MAIN, and ΔV MAIN values.

Referring still to FIG. **7** and in accordance with a preferred embodiment, a uv area is computed for groups of three texels in which two of the three texels lie on a common ORTHO scan line and the third texel is on a different scan line. Texels **140**, **142**, and **146**, for example, define a uv area **156** bounded by lines **148**, **150**, **152**, **154**. Similarly, uv area **166** defines the area associated with texels **142**, **144**, **148** and is the area bounded by lines **158**, **154**, **162**, **164**. The size of the uv area is proportional to the distance between texels selected by the texture map engine **126** and is thus related to the amount of compression that is appropriate.

In accordance with a preferred embodiment, the graphics processor selects texels from two LOD's and blends the selected texels together using the uv area. The uv area is a measure of the amount of compression that is appropriate for each texel value. If the uv area is 4.0, an appropriate texel is selected from LOD1 with its 4:1 compression. Similarly, a texel is selected from LOD2 with its 16:1 compression if the uv area is 16.0. Finally, a texel from LOD3 with its 64:1 compression is selected if the uv area is 64.0. It should be recognized, however, that the uv area may include values other than 4.0, 16.0, and 64.0.

Referring now to FIG. **9**, the preferred technique for blending LOD's will now be described. FIG. **9** shows a graph in which uv area is plotted on the horizontal axis and LOD is shown on the vertical axis. The relationship between the LOD's and the uv area is shown by segmented line **200** which includes segments **202**, **204**, **206**, **208**, **210**, **212**, and **214**. For purposes of this disclosure, the term "blending" refers both to selecting a single texel from one texture map as well as averaging two or more texels from multiple texture maps.

Texels from two LOD's preferably are blended together in a weighted average based on the uv area value if the uv area is in non-flat segments **204**, **208**, and **212**. Thus, texels from LOD0 and LOD1 are averaged together when the uv area lies in segment **204**. Likewise, texels from LOD1 and LOD2 are averaged when uv area is in segment **208** and texels from LOD2 and LOD3 are averaged when uv area is in segment **208** and texels from LOD2 and LOD3 are averaged if the uv area lies in segment **212**. The averaging is based on a scale factor associated with the uv area value and calculated by the texture map engine **126**. The scale factor represents the percentage of the uv area range of each non-flat segment **204**, **208**, **212** corresponding to a uv area value. The graphics processor **120** calculates a scale factor associated with uv areas in segment **204** as:

$$scale_{01}=(|uv\ area|-2)/2 \quad (1)$$

Where the vertical lines "|" around uv area indicates the absolute value of the uv area is to be used in equation (1). Subtracting a value of 2 normalizes the segment **204** to uv area values in the range of 0 to 2, rather than 2 to 4, and dividing by 2 computes the location of the uv area within the normalized range and is expressed as a percentage of the total range, 2. For example, a uv area value of 3.0 is half way between 2.0 and 4.0 and, using equation (1), results in a scale**01** factor of 0.5.

Scale factors for segments **208** and **212** are similarly calculated using equations (2) and (3) below:

$$scale_{12}=(|uv\ area|-8)/8 \quad (2)$$

$$scale_{23}=(|uv\ area|-32)/32 \quad (3)$$

where $scale_{12}$ is a scale factor for uv areas within segment **208** and $scale_{23}$ is a scale factor for uv areas within segment **212**.

Once calculated, the graphics processor uses a scale factor to blend texels from two LOD's based on the scale factor. The scale factors represent weights used by the graphics processor **120** to computed weighted average of texels. The texture map engine **126** averages texels from LOD0 and LOD1 as:

$$texel_{01}=(texel_0)(1-scale_{01})+(texel_1)(scale_{01}) \quad (4)$$

where $texel_0$ refers to a texel from $LOD0_1$ refers to a texel from LOD_1, and $texel_{01}$ refers to the weighted average of $texel_0$ and $texel_1$. $Texel_{01}$ is thus used to render texture on a pixel in a polygon. Similarly, texels from LOD1 and LOD2 and LOD2 and LOD3 are averaged per equations (5) and (6):

$$texel_{12}=(texel_1)(1-scale_{12})+(texel_2)(scale_{12}) \quad (5)$$

$$texel_{23}=(texel_2)(1-scale_{23})+(texel_3)(scale_{23}) \quad (6)$$

where $texel_2$ and $texel_3$ represent texels from LOD2 and LOD3, respectively, $texel_{12}$ is the weighted average of $texel_1$ and $texel_2$, and $texel_3$ is the weighted average of $texel_2$ and $texel_3$. Graphics processor **120** thus uses $texels_{12}$ and $texels_{23}$ to render texture in a polygon for uv areas lying within segments **208**,**212**.

The uv area values in FIG. **9** that mark the beginning and end points of the various segments are preferred, but other beginning and end points are acceptable. The beginning and end points shown in FIG. **9** were chosen to make calculating the scale factors simpler and faster. Dividing a number by (2n) where n is an integer is merely a bit shift operation by n bits and is easily performed by common shift registers. Similarly, subtraction by 2, 8, and 32 also is simple.

In flat segments **202**, **206**, **210**, and **214**, no averaging occurs for uv areas that fall within the uv area range associated with each of these flat segments. Thus, no averaging occurs if the uv area is in the range of 0 to 2.0, 4.0 to 8.0, 16.0 to 32.0, and greater than 64.0. Graphics processor **120** selects a texel from LOD0 if the uv area lies in flat segment **202**. Similarly, graphics processor selects a texel from LOD1 is the uv area is in segment **206**, a texel from LOD2 if the uv area is in segment **210**, and a texel from LOD3 if the uv area is in segment **214**. Alternatively stated, in flat segments **202**, **206**, **210**, and **214**, averaging does occur with a scale factor of 1.0. With a scale factor of 1.0, the "1–scale" terms in equations (5) and (6) will be 0 effectively eliminating one texel value from the averaging calculation.

Although numerous techniques known to those of ordinary skill in the art may be used to compute the uv areas, the following technique is preferred because of its simplicity and low computational overhead (few memory accesses and calculations). An arrow or vector is assigned to the texels as shown in FIG. **7**. Vector **128** begins at texel **140** and ends at

texel **142**. Vector **150** also begins at texel **140** and ends at texel **146**. The uv area **156** is determined from the "cross product" (also referred to as the "vector product") of vectors **148** and **150**. The cross product of two vectors (P and Q, for example) typically is represented as P×Q. The cross product of vectors P and Q results in a third vector V which points in a direction perpendicular to the plane defined by the two vectors P and Q (ie., into or out of the page). The magnitude or length of the resulting vector V is the product of the magnitudes of the vectors and the sine of the angle formed between the vectors. Thus, the uv area **156** can be calculated as the length of vector **148** times the length of vector **150** times angle **151**.

Referring to FIG. **8**, two vectors P and Q are shown represented by their rectangular components P_u and P_v and Q_u and Q_v. The magnitude of the cross product of the vectors P and Q may alternatively be calculated as:

$$V=P_xQ_y-P_yQ_x. \quad (7)$$

Where V is the magnitude of the cross product. Referring to FIG. **7**, the magnitude of the vector resulting from the cross product of vectors **148** and **150** (i.e., the uv area **156**) thus is calculated as:

$$\text{uv area}=(\Delta V \text{ MAIN}*\Delta U \text{ ORTHO})-(\Delta U \text{ MAIN}*\Delta V \text{ ORTHO}) \quad (8)$$

In accordance with a preferred embodiment, each uv area is calculated in software which may run in either the graphics processor **120** or CPU **102**. The following lines of code are exemplary of one way to calculate uv area (identified below as uv_area) in software.

```
xproda=(ΔV MAIN*ΔU ORTHO)&0x0000ffff;                    (9)

xprodb=(ΔU MAIN*ΔV ORTHO)&0x0000ffff;                    (10)

uv_area=xproda-xprodb;                                   (11)

if(uv_area&0x80000000)uv_area=-uv_area;                  (12)
```

In lines (3) and (4), the two terms in parentheses from equation (2) are calculated as xproda and xprodb, respectively. The term xprodb is subtracted from xproda in step (5). Finally, in step (6), the result of step (5) is examined to determine if it is a negative number. If so, the sign of uv_area is reversed to ensure uv_area is always a positive number. This step is commonly referred to as taking the absolute value of a number. It should be recognized that other techniques for taking the absolute value of a number are possible and consistent with the preferred embodiment.

Software may also be used to blend texel values together using the uv_area. Lines of code (13)–(124) below represent one embodiment of software to implement texel blending as described above.

```
if (uv_area < 0x00200)                                   (13)
  {                                                      (14)
    if (!lod0_npo2_en)                                   (15)
      lod = 0;                                           (16)
    lod_location = lod0_location;                        (17)
    lod_x = lod0_x_position;                             (18)
    lod_y = lod0_y_position;                             (19)
    lod_offset = lod0_tiled_offset;                      (20)
    if (!lod0_npo2_en)                                   (21)
      lod_s = 0;                                         (22)
    lod_location_s = lod0_location;                      (23)
    lod_x_s = lod0_x_position;                           (24)
    lod_y_s = lod0_y_position;                           (25)
    lod_offset_s = lod0_tiled_offset;                    (26)
    lod_scale = 0;                                       (27)
  }                                                      (28)
else if ((uv_area >= 0x00200) && (uv_area < 0x00400))    (29)
{                                                        (30)
  if(!lod0_npo2_en)                                      (31)
    lod = 0;                                             (32)
  lod_location = lod0_location;                          (33)
  lod_x = lod0_x_position;                               (34)
  lod_y = lod0_y_position;                               (35)
  lod_offset = lod0_tiled_offset;                        (36)
  if (!lod1_npo2_en)                                     (37)
    lod_s = 1;                                           (38)
  lod_location_s = lod1_location;                        (39)
  lod_x_s = lod1_x_position;                             (40)
  lod_y_s = lod1_y_position;                             (41)
  lod_offset_s = lod1_tiled_offset;                      (42)
  lod scale = (uv_area >> 1) & 0x0ff;                    (43)
}                                                        (44)
else if ((uv_area >= 0x00400) && (uv_area < 0x00800))    (45)
{                                                        (46)
  if(!lod1_npo2_en)                                      (47)
    lod = 1;                                             (48)
  lod_location = lod1_location;                          (49)
  lod_x = lod1_x_position;                               (50)
  lod_y = lod1_y_position;                               (51)
  lod_offset = lod1_tiled_offset;                        (52)
  if (!lod1_npo2_en)                                     (53)
    lod_s = 1                                            (54)
  lod_location_s = lod1_location;                        (55)
  lod_x_s = lod1_x_position;                             (56)
  lod_y_s = lod1_y_position;                             (57)
  lod_offset_s = lod1_tiled_offset;                      (58)
  lod_scale = 0;                                         (59)
}                                                        (60)
else if ((uv_area >= 0x00800) && (uv_area < 0x01000))    (61)
{                                                        (62)
  if (!l1od1_npo2_en)                                    (63)
    lod = 1;                                             (64)
  lod_location = lod1_location;                          (65)
  lod_x = lod1_x_position;                               (66)
  lod_y = lod1_y_position;                               (67)
  lod_offset = lod1_tiled_offset;                        (68)
  if (!lod2_npo2_en)                                     (69)
    lod_s = 2                                            (70)
  lod_location_s = lod2_location;                        (71)
  lod_x_s = lod2_x_position;                             (72)
  lod_y_s = lod2_y_position;                             (73)
  lod_offset_s = lod2_tiled_offset;                      (74)
  lod_scale = (uv_area >> 3) & 0x0ff;                    (75)
}                                                        (76)
else if ((uv_area >= 0x01000) && (uv_area < 0x02000))    (77)
{                                                        (78)
  if (!lod2_npo2_en)                                     (79)
    lod = 2;                                             (80)
  lod_location = lod2_location;                          (81)
  lod_x = lod2_x_position;                               (82)
  lod_y = lod2_y_position;                               (83)
  lod_offset = lod2_tiled_offset;                        (84)
  if (!lod2_npo2_en)                                     (85)
    lod_s = 2;                                           (86)
  lod_location_s = lod2_location;                        (87)
  lod_x_s = lod2_x_position;                             (88)
  lod_y_s = lod2_y_position;                             (89)
  lod_offset_s = lod2_tiled_offset;                      (90)
    lod_scale = 0;                                       (91)
}                                                        (92)
else if ((uv_area >= 0x02000) && (uv_area < 0x04000))    (93)
{                                                        (94)
  if (!lod2_npo2_en)                                     (95)
    lod = 2;                                             (96)
  lod_location = lod2_location;                          (97)
  lod_x = lod2_x_position;                               (98)
  lod_y = lod2_y_position;                               (99)
  lod_offset = lod2_tiled_offset;                       (100)
  if (!lod3_npo2_en)                                    (101)
  lod_s = 3;                                            (102)
  lod_location_s = lod3_location;                       (103)
  iod_x_s = lod3_x_position;                            (104)
  lod_y_s = lod3_y_position;                            (105)
  lod_offset_s = lod3_tiled_offset;                     (106)
```

-continued

```
    lod_scale = (uv_area >> 5) & 0x0ff;                    (107)
  }                                                        (108)
  else if (uv_area >= 0x04000)                             (109)
  {                                                        (110)
    if (!lod3_npo2_en)                                     (111)
      lod = 3;                                             (112)
    lod_location = lod3_location;                          (113)
    lod_x = lod3_x_position;                               (114)
    lod_y = lod3_y_position;                               (115)
    lod_offset = lod3_tiled_offset;                        (116)
    if (!lod3_npo2_en)                                     (117)
      lod_s = 3;                                           (118)
    lod_location_s = lod3_location                         (119)
    lod_x_s = lod3_x_position;                             (120)
    lod_y_s = lod3_y_position;                             (121)
    lod_offset_s = lod3_tiled_offset;                      (122)
    lod_scale = 0;                                         (123)
  }                                                        (124)
```

With reference to lines of code (13)–(124) and FIG. **9**, code line (13) determines whether the uv_area is less than **2** (ie., portion **202** in FIG. **9**). Similarly, in code line (29), the software determines whether the uv_area is greater than **2** and less than **4**. This range of uv area corresponds to portion **204** in FIG. **9**. In code line (45), the software determines whether the uv_area is greater than or equal to 4 and less than 8, corresponding to portion **206** in FIG. **9**. In code line (61) the software determines whether the uv_area is between 8 and 16 (portion **208**). Code line (77) determines whether the uv area is greater than or equal to 16 and less than 32 (portion **210**). Finally code lines (93) and (109) determine whether the uv area is greater than or equal to 32 and less than 64 (portion **212**) or greater than or equal to 64 (portion **214**), respectively. It should be noted that in comparison steps (13), (29), (45), (61), (77), (93), and (109) the comparator operator's "greater than or equal to" and "less than" have been used. It should be recognized, however, that other comparator operators, such as "greater than" and "less than or equal to" could also be used. Further, the range of values to which the uv_area is compared can be different than that shown in the code listing above.

The software preferably converts the uv_area to an lod_scale value representing the fraction of a uv area range corresponding to the particular uv_area value. This conversion process is preferably accomplished in software by shifting the uv_area value to the right by one or more bit positions and logically ANDing the shifted result with the value FF hexadecimal. For instance, in code line (43) the uv_area value is shifted to the right by one bit and that result is ANDed with FF. Conversion step (43) is performed when the uv_area is greater than 2 and less than 4 as determined by code step (29). The resulting lod_scale value corresponds to the percentage of the range between 2 and 4 to which the uv_area value corresponds.

Similarly, in code step (75) the uv-area value is converted to a percentage between 8 and 16 (code step (61)) by shifting the uv-area by three bit positions to the right and then ANDing that result by zero FF hexadecimal. Finally, in code step (107) the uv_area value is bit shifted to the right by five bit positions and than ANDded with FF to produce an lod_scale value that corresponds to the percentage of the range between 32 and 64 to which the uv_area value corresponds. Thus code steps (43), (75), and (107) convert the uv_area value for the ranges **204**, **208**, and **212** in FIG. **9**.

If the uv_area values, however, correspond to flat portions **202**, **206**, and **210** in FIG. **9**, the lod_scale value is set to zero indicating that no averaging is to occur. Setting the lod_scale value to zero is accomplished in code steps (27), (59), (90), and (123).

The remaining lines of code shown above generally represent the initialization of pointers to the various texture maps from which the texel values will be selected for blending in accordance the preferred embodiment above.

In the following lines of code (125)–(127), the software calculates a blended texel value based on texels selected from two different texture maps and the lod_scale value calculated above. For an lod_scale value of zero, frac_munge_2 is zero and frac_munge_1 is the difference between the value 100 hexadecimal and the lod_scale value. Accordingly, the tex_mask value in step (127) is the texel value selected from one texture map. For a non-zero lod_scale value, the frac_munge 1 and frac_munge_2 values will be non zero and thus the tex_mask value will represent the blending of two texel values in accordance with the preferred embodiments described above.

```
frac_munge_1 = 0x100 - lod_scale;                          (125)

frac_munge_2 = lod_scale;                                  (126)

tex_mask = (tex_mask_local[0] * frac_munge_1) +            (127)
           (tex_mask_local[1] * frac_munge_2);
```

Numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. It is intended that the following claims be interpreted to embrace all such variations and modifications.

We claim:

1. A texture mapping method comprising the steps of:

calculating texel coordinates:

calculating an area bounded by a plurality of texel coordinates;

comparing said area to a plurality of ranges of areas, said plurality of ranges of areas including a first subset of ranges of areas and a second subset of ranges of areas; and

blending at least one texel value from a texture map based on said area by calculating a weighted average of at least two texel values selected from different texture maps when said area corresponds to said first subset and calculating a scale factor based on said area,

wherein said scale factor is greater than or equal to zero.

2. The texture mapping method of claim **1** wherein said step of calculating said scale factor includes dividing said area by m wherein m corresponds to a range of areas in said first subset.

3. The texture mapping method of claim **2** wherein said area comprises a multi-bit digital value and said step of dividing said area by m includes shifting the bits of said area by n bit positions to the right to produce a shifted value.

4. The texture mapping method of claim **3** wherein said first subset includes a first range, a second range, and a third range of areas and n is 1 for said first range.

5. A computer readable storage medium for storing an executable set of software instructions which, when inserted into a host computer system, is capable of controlling operation of the host computer, said software instructions being operable to blend at least one texel value selected form a plurality of texture maps, said computer readable storage medium comprising:

an instruction for calculating texel coordinates;

an instruction for calculating an area bounded by a plurality of texel coordinates;

an instruction for comparing said area to a plurality of ranges of areas, where said plurality of ranges of areas includes a first subset of ranges of areas and a second subset of ranges of areas; and

an instruction for blending said at least one texel value based on said area by calculating a weighted average of at least two texel values selected from different texture maps when said area corresponds to said first subset, and calculating a scale factor based on said area,

wherein said scale factor is greater than or equal to zero.

6. The computer readable storage medium of claim 5 wherein said instruction for calculating said scale factor includes an instruction for dividing said area by m, wherein m corresponds to a range of areas in said first subset.

7. The computer readable storage medium of claim 6 wherein said area comprises a multi-bit digital value and said instruction for dividing said area by m includes an instruction for shifting the bits of said area by n bit positions to the right to produce a shifted value.

8. The computer readable storage medium of claim 7 wherein the first subset includes a first range, a second range, and a third range of areas and n is 1 for said first range.

9. The computer readable storage medium of claim 8 where n is 3 for said second range and n is 5 for said third range.

10. The computer readable storage medium of claim 9 further including an instruction for logically ANDing said shifted value with a FF hexadecimal value.

11. A texture mapping method, comprising the steps of:

calculating texel coordinates calculating an area bounded by a plurality of texel coordinates;

blending at least one texel value from a texture map based on said area, said blending comprising:

comparing said area to a plurality of ranges comprising a first subset of ranges of areas and a second subset of ranges of areas; and

calculating a weighted average of at least two texel values selected from different texture maps when said area corresponds to said first subset by calculating a scale factor by dividing said area by m wherein m corresponds to a range of areas in said first subset, and wherein said sale factor is greater than or equal to zero; and

said area comprises a multi-bit digital value and dividing said area by m includes shifting the bits of said area by n bit positions to the right to produce a shifted value, and wherein said first subset includes a first range, a second range, and a third range of areas and n is 1 for said first range, n is 3 for said second range, and n is 5 for said third range.

12. The texture mapping method of claim 11 further including the step of logically ANDing said shifted value with a FF hexadecimal value.

13. A texture mapping method, comprising the steps of:

a) calculating texel coordinates based on a ΔU ORTHO value, a ΔV ORTHO value, a ΔU MAIN value, and a ΔV MAIN value;

b) calculating an area bounded by a plurality of texel coordinates by multiplying said ΔV MAIN value by said ΔU ORTHO value to produce a first product;

c) comparing said area to a plurality of ranges of area values;

d) calculating a scale factor based on said area;

e) selecting a first texel value from a first texture map;

f) selecting a second texel value from a second texture map; and

g) calculating a weighted average of said first and second texel values using said scale factor.

14. The texture mapping method of claim 13, wherein step (b) further includes the step of multiplying said ΔU MAIN value by said ΔV ORTHO value to produce a second product.

15. The texture mapping method of claim 14 wherein step (b) further includes the step of calculating a difference between said first and second products.

16. A texture mapping method, comprising the steps of:

calculating texel coordinates based on a ΔU ORTHO value, a ΔV ORTHO value, a ΔU MAIN value, and a ΔV MAIN value;

calculating an area bounded by a plurality of texel coordinates;

comparing said area to a plurality of ranges of area values;

calculating a scale factor based on said area;

selecting a first texel value from a first texture map;

selecting a second texel value from a second texture map; and

calculating a weighted average of said first and second texel values using said scale factor.

* * * * *